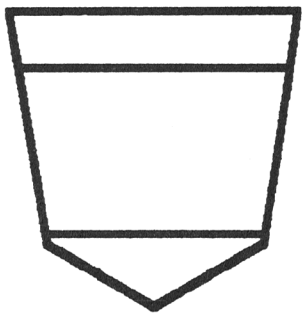
POCKET
COMPUTER
NEWSLETTER



(c) Copyright 1983 POCKET COMPUTER NEWSLETTER

Issue 29 - November

PORTABLE COMPUTER TRAVELS LIGHT

MicroOffice Systems Technology has announced a portable computer designed for travelers. The five pound unit can serve in the office and on the road as a remote terminal or stand-alone personal computer.

The machine has been given the trademarked name "RoadRunner." It weighs just five pounds, operates from self-contained rechargeable batteries, and is notebook-sized. It measures a trim 7-3/4 by 11-1/2 by 3 inches. At this size it fits in one-half of a standard business briefcase. The other half of the briefcase can be used for the invariable array of documents that typically accompany the harried traveler. The company believes the RoadRunner is ideal for use by writers, salespeople and others who need a small, yet powerful, portable office work station.

Using appropriate software modules, the unit permits users to create account files, write letters and reports, analyze financial models, track expenses, follow orders and proposals, and access remote databases, if desired.

The mobile unit features a standard-sized 73-key keyboard in the usual QWERTY format. There are 18 functions keys. Eight of these are used to select operations from a menu that can be shown on the system's built-in LCD screen.

In fact, the 8-line by 80-column LCD screen is built as part of the unit's cover. The cover flips up to automatically turn the computer on and provide a convenient viewing angle of the screen. When this cover is closed, the system's keyboard and plug-in software modules are fully protected from accidental damage in a traveling environment.

A series of plug-in modules provide a wide range of capabilities. Modules may contain up to 32 kilobytes of RAM or 128 kilobytes of ROM. The modules are said to provide advantages over the use of sensitive disk drives in portable situations.

The unit utilizes a RS-232C interface for communicating with external devices. An internal 300 baud auto-dial, auto-answer modem may be used for communicating with other terminals or for utilizing remote computers and databases.

The RoadRunner can be operated for more than eight hours before its battery pack is exhausted.

Software packages to be made available for the machine include: a text editor, spreadsheet, BASIC, phone list, appointment manager and terminal communications package. The company also plans to provide a variety of CP/M-compatible programs through its plug-in modules.

The single-quantity price for RoadRunner ranges from $1,600 - $2,000 depending on options selected. Productions units are scheduled for delivery in late November, 1983.

The RoadRunner has been under development since the company's founding in late 1981. The firm, MicroOffice Systems Technology, Inc., was founded as an organization dedicated to the development, manufacture and marketing of advanced portable information products.

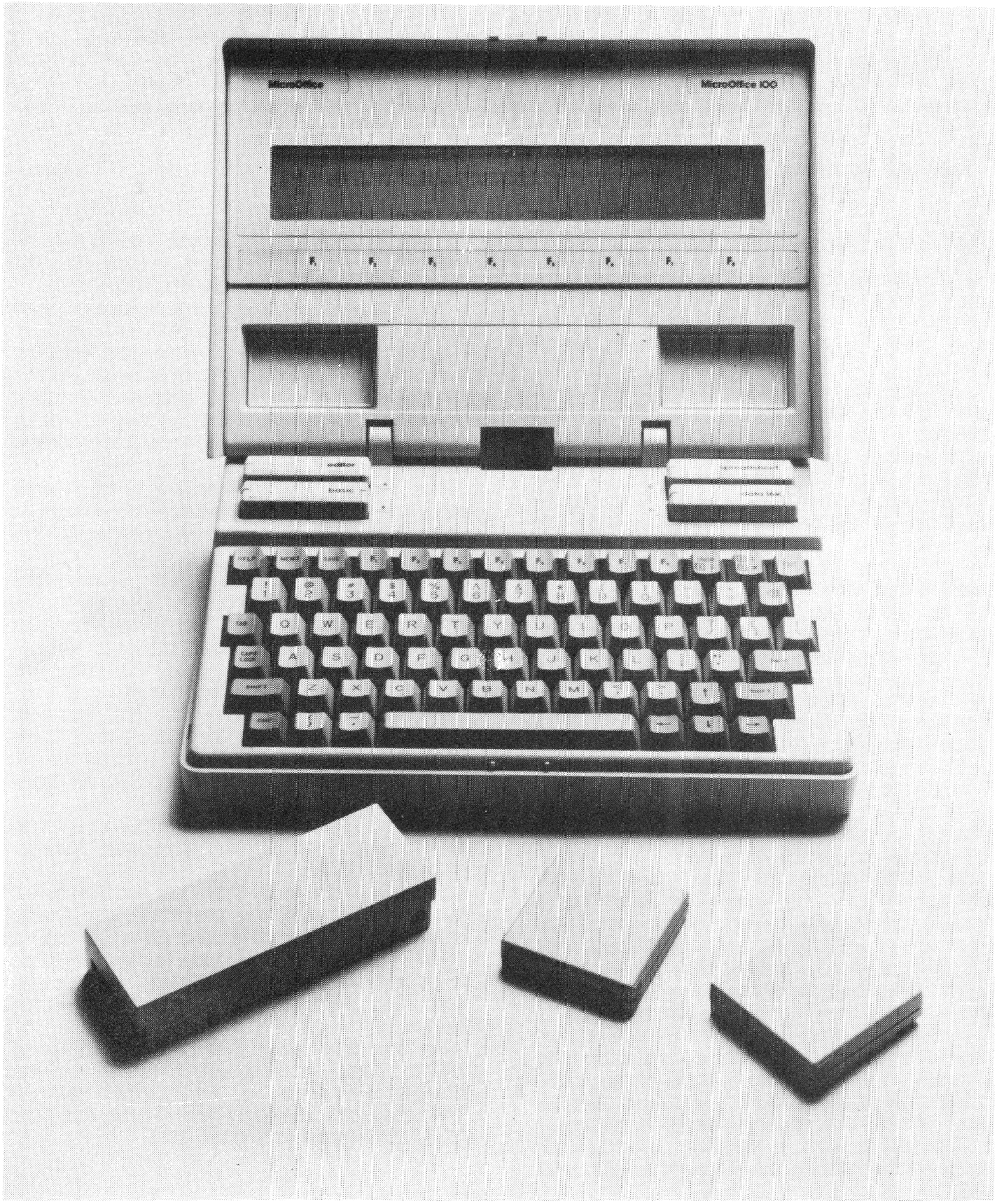
The company is headed by Robert F. Weltzein who serves as Chairman of the Board and CEO. He served formerly as President of Citizen Watch. Prior to that, he was Chairman and CEO of Timex Corporation.

The firm's president is James P. Dunn. Mr. Dunn was formerly with Exxon Enterprises, Hendrix Electronics and IBM.

John W. Chapman is Secretary and Treasurer. Mr. Chapman is also Vice President and General Counsel for the Computer and Communications Industry Association. Previously he has served on the corporate staff of Xerox, Honeywell Information Systems and California Computer Products.

For additonal information on the RoadRunner portable computer, contact: *Marvin Preston, Director of Marketing, MicroOffice Systems Technology, Inc., 35 Kings Highway East, Fairfield, CT 06430.* Telephone (203) 367-2525.

Photo *The RoadRunner portable computer is shown with its removable battery pack and several of its plug-in modules. Up to four modules may be installed at one time. The flip-up cover contains the display and provides protection for the keyboard when closed.*



THE CASIO FX-700P

At first feel, the FX-700P lies somewhere between a Sharp PC-1250 and the earlier Radio Shack PC-1 in size. That is, it is certainly bigger than the 1250, but slimmer, shorter and not as "chunky" feeling as the PC-1. It does, indeed, slip into a shirt pocket and is light enough (at just 4.2 ounces) so that one feels comfortable with it there.

There is more room for the keys on the Casio 700P than there is on the Sharp PC-1250. They are larger and spaced further apart. Thus, keying in programs is not as much of a chore as it sometimes feels when keying in a large program on the tiny keys of the 1250.

I would say that the 700P might have special appeal to engineers and scientists. For one thing, it sports a good selection of mathematical functions. These include: SIN, COS, TAN, ASN, ACS, ATN, LOG, LN, EXP, SQR, ABS, SGN, INT, FRAC and RAN#. All of these are available at the stroke of two keys: a special (F) function key and the key that is sublabeled with the corresponding legend. For another, most of the commonly used BASIC statements are also available for two keystrokes. The keywords GOSUB, FOR, TO, STEP, NEXT, GOTO, IF, THEN, PRINT, RETURN, STOP, END, DEFM and INPUT, as well as the commands LIST and RUN, are easily brought to the display. Just press the special shift (S) key and the appropriately superlabeled key. This feature can be particularly nice if you are not accustomed to doing a lot of typing on a QWERTY keyboard, which is the layout used on this machine.

Another factor in favor of the FX-700P if you plan to do a fair amount of number crunching, is its relative speed. Based on a simple loop test, it is more than twice as fast as its predecessor (the FX-702P). In another comparison using a simple benchmark that loops, calls a subroutine and performs standard mathematical operations, the little FX-700P beat the PC-1250 by 2:1 and was virtually tied with the PC-1500. Quite impressive.

At first I was a little dubious about the practicality of having just a 12-character LCD display. Experience has shown that, at least in this implementation, it is acceptable. Numbers may be displayed in 10-digt signed mantissa format or an 8-digit mantissa with 2-digit exponent. One reason Casio can get away with such a short display is the wise manner in which they implemented its operation. When displaying a long message, such as via a PRINT statement, the screen automatically scrolls at a comfortable rate.

The little LCD can depict a full complement of alphanumeric characters. In addition to uppercase representations, a special EXTension mode allows lowercase characters and even a variety of special mathematical symbols and a few graphics (such as representations of card suits: the spade, heart, diamond and club symbols) to be displayed.

The 700P seems especially well-suited for people who need to repetitively perform a number of relatively limited procedures. You can store up to ten different programs in its memory at one time. Each program area (designated P0 - P9) is accessed independently (even though all programs are sharing the same block of user memory). Thus, you can re-use the same group of line numbers in each program, if desired.

It is as if you had 10 separate little PCs built into one keyboard/display package. Each program area can be edited or executed at will. You can allocate the amount of memory assigned to each program from amongst the total of 1568 program steps (or 222 variables). If you divided this into equal portions , each program area could contain about 156 program steps (bytes). Put your ten most frequently used programs in memory and solve at will!

The BASIC interpreter is similar in overall capabilities to most other PCs. There are some special features that deal with the partitioning capability of the machine. For instance, the use of the statement GOSUB #1 enables a user to call a complete program module as a subroutine. While the machine does not provide for the use of labels on program lines, GOTO and GOSUB statements can use variable values to represent line numbers.

As might be expected on a machine having only a 12-character display, string capabilities are a bit limited: about on a par with a Radio Shack PC-1. Except for one special 30-character ($) string storage area, standard string variables are limited to just seven characters in length. However, the Casio FX-700P can test for greater than/less than when processing string variables. The PC-1 could only check for equality.

The 700P does not have any audio capability. If you are going to let the PC crank away on projects that take a lot of time, you will have to periodically check the display for results. You cannot generate a cheery little beep to alert you when the task has been completed.

Still, all-in-all, it is a nice, slim, comfortable, light-weight, fully-functional PC. At a U.S. list price of $99.95, it is competitive with other brands.

Programs can be saved to or restored from an audio cassette recorder using an optional recorder interface dubbed the FA-3. This item has a U.S. list price of $39.95. An optional thermal printer, the FP-12, at $69.95, provides capability for obtaining hardcopy of programs and data.

FUEL CONSUMPTION ANALYST

R. M. Organ, 9501 Nowell Drive, Bethesda, MD 20817, who describes himself as an "absolute beginner" to the world of programming, has come up with this analyzer for the fuel-conscious motorist. You can use it to generate a chart that can reveal the influence on fuel consumption caused by such factors as highway versus city driving, winter versus summer, one driver versus another, the consequences of mechanical defects such as brakes that rub or fuel swilling out of a loose cap on the gas tank, etc.

You use the program by transferring data from a logbook. This book should record the number of gallons put in and the odometer mileage at each fill-up. As each pair of entries is inputted, the plotter unit draws a bar representing the miles-per-gallon achieved from the previous tankful. Successive entries result in more bars. A graph builds up as paper rolls through. The program automatically extends divisions on the MPG scale as needed and periodically repeats the MPG scale. You can build a chart that covers 100,000 miles or more, if you don't mind two feet (plus) of paper! When all of the desired data has been fed in, the program does still more: it prints a table of frequencies, listing the number of occurrences of each integer MPG, and plots a histogram. This makes it easy to spot which MPG occurs most often or determine whether there are several peaks. Those might represent the results of two drivers, different seasons of the year, etc.? The user is also given the option of having a summary of overall average MPG calculated for the entire period covered by the chart.

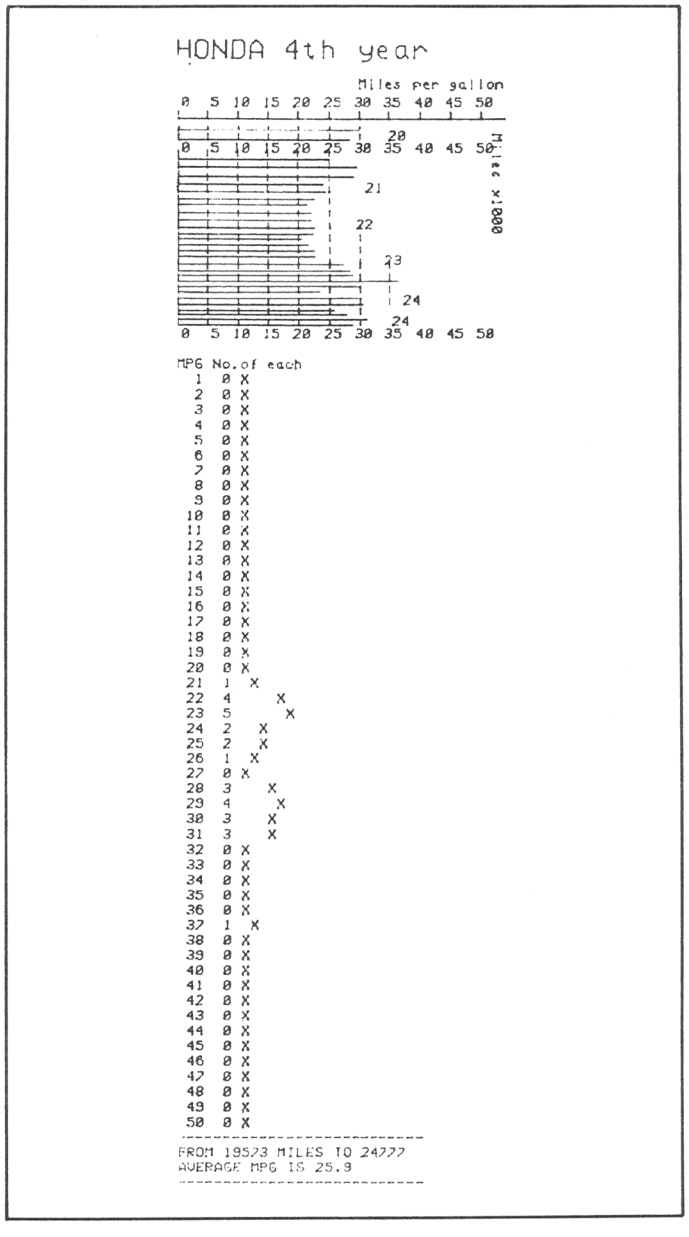
Overview of the Program

The program will execute in an unexpanded Radio Shack PC-2 or Sharp PC-1500. It does a lot of work for the user. After the user has entered the EPA estimated MPG, the program selects its own scale for the MPG axis. It averages the MPG from successive entries, thereby taking care of the times when the tank was underfilled/overfilled, which could produce an unsatisfactory appearance on the chart. It shows the MPG as closely as the plotter allows. This is one two-hundredth of the maximum MPG value shown on the scale. For a 30 MPG scale, it would be shown to within 0.15 MPG. (Of course, calculations maintain a higher degree of accuracy.) The histogram value is rounded to the *nearest* whole number, instead of always being rounded downwards. The MPG scale will accept any maximum value in the range 10 to 100 in tens. The program stops if any figure over 99 (MPG) is produced. This is an unlikely event. The program also has other safeguards.

Tips On Inputting Data

Make sure the entries in your log are legible, do not have gross errors, and are in proper sequence *before* you commence inputting to the computer. It is a good idea to lay a ruler or piece of paper beneath the entry line you are reading to help keep track of your place and coordinate gallon and mileage data pairs. Check the LCD after each entry to verify proper keying *before* you press the ENTER key. Wait for the computer to generate its beeps before starting to input the next data item. The computer and printer sometimes have quite a bit to do between entries. If a premature entry is attempted, the program may halt. It is also advisable to cover up the arithmetic operator keys (/ to +) or take great care not to inadvertently press those buttons while inputting numbers. Failure to

Example *Chart Produced by the Fuel Program.*



heed this advice may bring the operation of the program to an untimely halt.

Program Description

The program runs in several successive phases as indicated by the accompanying flowchart. The first phase accepts the name of a car plus notes to a total of 16 characters.

Line 710 zeros some important variables and prompts for the maximum MPG expected. Lines 720 and 730 round this answer up to the next largest decade as the maximum scale value. Line 740 calculates distances for the pen to move and the numbers to be printed on the axis. Lines 750 to 780 draw and print. Line 790 is used to title the mileage axis (which will develop as the program proceeds). Line 795 dimensions an array (that will be used by the histogram later) and fills it with zeros.

Line 800 accepts the first pair of entries. One of these is remembered for use in the final summary. The other is set to zero since it will not be needed in the first calculation but it is convenient to establish the habit of keying in pairs. Line 805 foreshadows serious operations: O, M and N are variables to contain mileage. N will always hold the most recently entered, M the earlier and O the one before. Thus, for every new entry its two predecessors must have been moved back one along the series. Later, in line 830, the existence of O will enable the program to average the mileage from two fills.

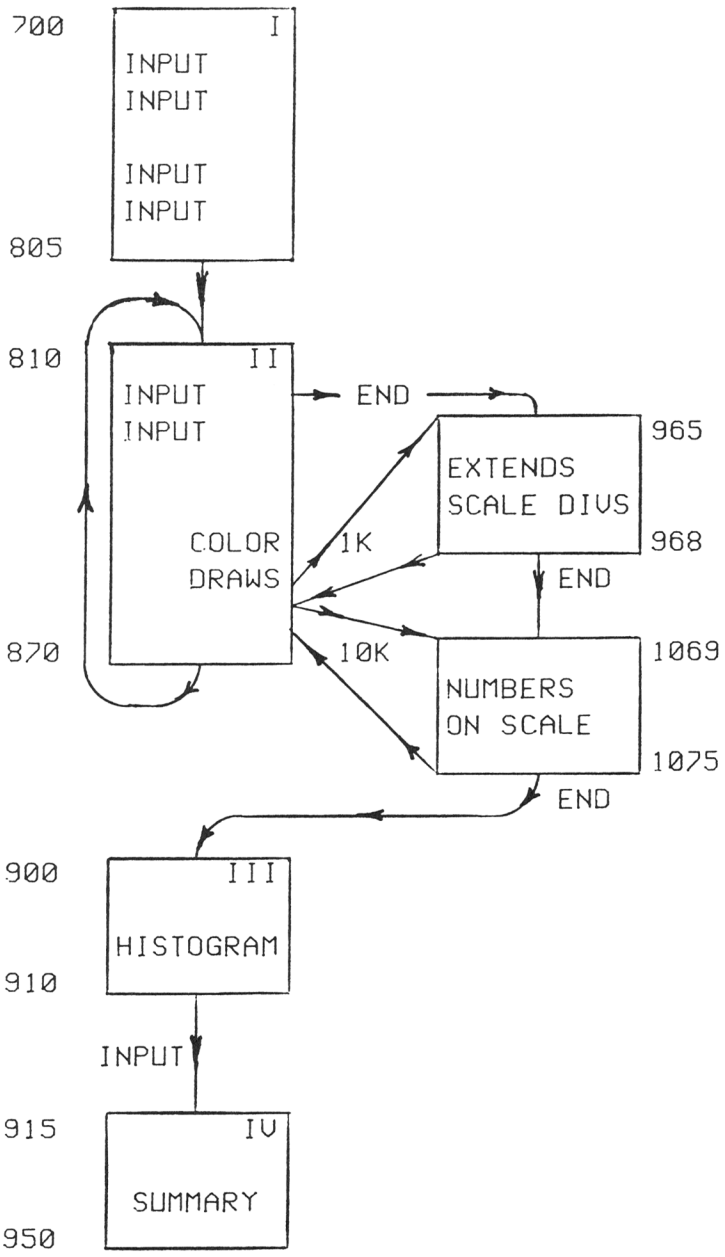
J is a variable paired with I which contains the number of ten-thousands of miles. At the end of the second phase J will become I and S will become R. However, they are each calculated here in order to avoid having extra scales printed immediately next to the axis, as would otherwise happen if the second mileage to be entered introduced a new thousand or ten-thousand. This line, 805, ends the first phase of the program, which is used only once.

The next phase is repeated for every pair of entries. Its objectives are: to draw a bar of the correct length, to test whether the newest entry has changed its number into the next thousand or ten thousand, to begin an ending-off operation if the prompt to "end" has been responded to affirmatively, or to return control back to its beginning in order to repeat the loop.

Line 810 accepts the next entry for mileage or an indication to end. Line 820 accepts an entry for gallons and adds it to an accumulating total. Lines 830 and 840 make a number of calculations: notably average MPG, the distance the printer pen must move for this and for miles run, the number of thousands and ten thousands in the new mileage, of MPG as an integer, and pen color.

Color was changed each thousand miles as a way of avoiding construction of the usual miles axis. This would have taken up much needed space at the edge of the narrow paper tape. Considerable thought was needed to find a formula that would convert successive thousands into a repeating sequence of 0 to 3. In line 850 that particular MPG in the array that corresponds with the MPG just calculated is increased in number by one. The printer is prepared to draw the bar and the actual printing takes place in line 860. Lines 863 and 865 test whether the mileage has increased into a new thousand or ten-thousand. If so, direct control is passed to the appropriate subroutines starting in lines 965 and 1069.

Flowchart *Operation of the Fuel Consumption Program.*



The final line of this phase is number 870. It moves numbers back along their storage sequences, leaving variables free to accept new entries, then loops back to the start of this phase.

Line 900 starts a third phase which constructs a histogram as the first part of an end-off sequence. In the text mode, two columns are headed, one for the name of MPG and the other for how many of each MPG occur. The baseline for the histogram is defined as the seventh column from the edge of the printout. Lines 905 and 910 print the data stored in the array into the selected format to complete phase three.

Line 915 begins with a prompt for the second part of the end-off, which is phase four. Intellectually, this is the least satisfying part of the program, as it ought to continue automatically. Every attempt to cause this failed because various PRINT USING statements are included in the earlier phases for good reasons. These are not appropriate to the format used in phase four. The only way that this programmer could find to eliminate memory of the earlier PRINT USINGs was to include an instruction to RUN. Apparently this clears all variables except those designated by letters, which, by great fortune, are those used here.

[The use of a USING statement without any parameters will return the PC to its normal "unformatted" state of operation. -Editor.]

Line 920 formats the calculated overall MPG into a single decimal place. It was invented to avoid use of the PRINT USING statement, which would have clouded the running of lines 930 and 935. These define strings and then print them with numbers interpolated. It is the fact that these numbers can vary in length from two-digits to six-digits, dependent on the miles entered, that causes

Program *Charting Fuel Consumption.*

```
700:"FUELUSE":WAIT 130:PRINT "FUELUSE by R.M.ORGAN":CLEAR
705:INPUT "NAME OF CAR? ";E$:CSIZE 2:LPRINT E$
710:TEXT :CSIZE 1:LCURSOR 20:LPRINT "Miles per gallon":LET B=0,D=0,I=0,J=0,N=0,Q=0,S=0
715:LET W=0,Z=0:LF 1:INPUT "EPA MPG (Hwy) is ";E
720:IF (E/10-INT (E/10))=0THEN GOTO 740
730:IF (E/10-INT (E/10))<>0THEN LET E=E+1:GOTO 720
740:GRAPH :T=20:FOR X=0TO 200 STEP T:Z=X/200*E
750:LINE -(X,0)-(X,5),0,0
760:GLCURSOR ((X-15),8):CSIZE 1:LPRINT USING "####";Z:GLCURSOR (X,0)
770:NEXT X
780:LINE -(220,0),0,0
790:ROTATE 1:GLCURSOR (207,-10):LPRINT "Miles x1000":GLCURSOR (0,-5):ROTATE 0
795:DIM A(E):FOR H=0TO E:LET A(H)=0:NEXT H
800:TEXT :BEEP 1,8,50:INPUT "MILEAGE=";M:BEEP 2,8,50:INPUT "GALLONS=";K:LET B=M,K=0
805:LET O=M:LET J=INT (M/10000):LET S=INT (M/1000)
810:BEEP 1,8,50:INPUT "NEXT MILEAGE(1 for END)=";N:IF N=1 THEN 965
820:BEEP 3,8,50:INPUT "NEXT GALLONS=";L:LET W=W+L
830:LET U=(N-O):LET G=U/(K+L),X=G*200/E:LET Y=U/75:LET Q=Q+Y:LET R=INT (N/1000)
840:LET I=INT (N/10000):LET C=INT (4*(N/4000-INT (N/4000))):LET H=INT (G+0.5)
850:LET A(H)=A(H)+1:GRAPH :CSIZE 1
860:LINE -(0,-Y)-(X,-Y)-(0,-Y),0,C:GLCURSOR (0,-Y):SORGN
863:IF S<>RGOSUB 965
865:IF J<>IGOSUB 1069
870:LET O=M,M=N,K=L,J=I:IF N<>1 THEN 810
900:TEXT :CSIZE 1:COLOR 0:LF 2:LPRINT "MPG No. of each";FOR H=1TO E:LET F=A(H)+7
905:LPRINT USING "###";H;A(H)
910:LF -1:LCURSOR F:LPRINT "X":NEXT H
915:BEEP 5,210,100:WAIT :PRINT "Now RUN 920 or OFF"
920:LET Z=0.1*INT (10*(M-B)/W):COLOR 3:LPRINT "----------"
930:A$="FROM":B$=" MILES TO":C$="AVERAGE MPG IS":LPRINT A$;B;B$;M
935:LPRINT C$;Z
940:LPRINT "----------"
950:COLOR 0:LCURSOR 0:END
965:LET U=INT X/T:FOR P=0TO (U*T)STEP T:LINE (P,0)-(P,Q),3:NEXT P
966:GLCURSOR ((X+15),0):LPRINT R;GLCURSOR (0,0):SORGN :LET S=R,Q=0
967:IF N=1THEN 1069
968:RETURN
1069:FOR X=0TO 200STEP T:Z=X/200*E:GLCURSOR ((X-15),-8):LPRINT USING "####";Z:NEXT X
1070:IF N<>1GOTO 870
1075:IF N=1GOTO 900
```

a fixed PRINT USING statement to be unsatisfactory unless one is prepared to accept unsightly variable numbers of spaces in a sentence. Line 950 brings the program to an end.

The subroutine contained in lines 965 to 968 serves to put dotted lines across the bars at intervals (T) corresponding with the intervals between divisions on the MPG axis. The lengths of these lines usually represents 1000 miles, because they are added every time the pen changes color. However, the program accumulates the actual distance since the pen last changed, in variable Q. This is because the run can be terminated at any mileage, probably less than 1000, and part of the end-off operation is to extend these dotted lines. Line 966 moves the pen to a position 15 units beyond the tip of the bar. The number of thousand miles is printed where it is usually clearly visible on a white background.

After the printing has been done, the various variables involved are either moved back along the series (S) or set to zero again (Q) ready to be used in the next series.

The subroutine in lines 1069 and 1070 repeats the printing of numbers used for the MPG axis in lines 740 to 760. However, it is done without the associated plain line. The string of numbers is printed every time the mileage changes into a new ten-thousand. An associated plain line might be confused with a MPG bar.

Use of Variables

Here is a list of the variables used in the program:

A is the title for the array of integers for MPG used in the histogram.

B is the store for the very first entry of mileage, called upon in phase four.

C is for the color of the pen.

E begins as the EPA estimate of MPG but is raised by the program to the length of the MPG scale.

F is the tab value of the X that is printed at the end of each histogram bar.

G is the MPG calculated from the most recent two pairs of entries.

H is the integer MPG calculated for use in the histogram.

I is the number of ten-thousand miles calculated for the current entry.

J is the I of the previous run through phase two.

K is the number of gallons used in the first tank filling. On later runs this becomes the stored value of the preceding filling.

L is the current number of gallons put in the tank.

M is the mileage.

N is the next mileage to be entered.

O is a store for M so that M can store N when a new N is to be entered.

P is the number of printer units between dotted lines of the scale markers.

Q is the number of printer units that have accumulated along the miles axis since the pen last changed color.

R is the newly calculated ineger number of thousand miles.

S is a store for the value of R calculated for the previous bar.

T is the number of printer steps in each division of the MPG axis.

U is the number of divisions of the MPG axis included in the length of the previous bar. The bars vary in length. In order not to clutter the chart in the area where the miles numbers will be printed, the dotted lines are only printed where needed, up to division U.

V is the mileage run between fills.

W is used to accumulate the number of gallons used since the first entry.

X is the distance in printer units that the pen moves parallel with the MPG axis.

Y is the distance in printer units that the pen moves parallel with the miles axis.

Z is a number, wherever it may be printed.

Possible Alterations

For those whoe would like to experiment, there are some improvements that could be made.

The number 75 in line 830 can be changed to 50 or 100 in order to expand or contract the miles axis.

If additional memory is available, the following lines may be added:

```
845 ON ERROR GOTO 1080
1080 WAIT:PRINT "RESTART- ADD 10 TO EPA MPG"
```

These changes will prompt action if an excessive MPG turns up and stops the run. The extra memory needed to accomodate this addition will be available already if the highest MPG does not exceed 30. However, if you should then have to take the advice and raise MPG to 40, there may be too little memory left for the array.

If your Radio Shack PC-2 accepts the statement DIM A(E)*2 in line 795, as the insert in the manual suggests it should (but mine does not), then the total memory needed will shrink considerably and should find few restrictions. The array cannot be made exactly the right size because it must accept unusually large and small MPG that happen sporadically, otherwise the program will stop.

[Thanks, R. M. Organ, for showing us how a beginning programmer tackles a less than trivial programming project. -Editor]

ALTERNATE CONTROL OF THE PC-1500 LCD

This is the second part of the discussion on this subject matter that was started in Issue 28 of *PCN*. You may want to refer to that article as I will be picking up where I left off. At that point I had presented a general line drawing routine in BASIC and a few supporting routines. These were used to demonstrate that a line could be drawn between any two points on the LCD. I suggested, at the close of the article, that those interested might find it challenging to work on converting the pixel control routine (labeled ON) into machine language. This would enable the drawing of a line to occur at a faster rate. Did you work on the problem?

Creating A Hybrid Program

Mixing machine language and BASIC routines (or any high level language routines) results in what I like to refer to as a *hybrid* program. The use of this technique on the PC-1500 has special advantages. For one thing, it makes it easy to control, for example, input operations -- which could get pretty hairy trying to do in machine language -- as these operations can be left in BASIC. But, when you want to maximize the performance in some area -- such as we are working on doing with the LCD graphics -- then you can go right in and tweak things up using super-fast MLP techniques.

The BASIC used in the PC-1500 makes it easy to construct hybrid programs. You can cross from BASIC to machine language by using the CALL statement. Using CALL &XXXX, where &XXXX represents an address in hexadecimal notation, will simply direct the PC to call a machine language routine at the given address as a subroutine. When a machine language return instruction (RTS) is found, program control will pass back to the BASIC program, just as though you had used a GOSUB statement.

If you want to pass a value from BASIC to a machine language routine, then you may use a variation of the CALL statement. The statement form CALL &XXXX,N will pass the value of variable N to the machine language routine. It does this by placing the variable value into the 16-bit X register of the CPU where it may be picked up and further manipulated by the machine language routine. There are, however, a few restrictions that must be observed: only integer values that fit within 15 bits plus a sign bit, (+32,767 to -32,768) may be passed. If you try to pass a non-integer value using this technique, the BASIC interpreter will effectively integerize it. If you try to pass a value outside this range, you will generate an error message.

You can also pass a value back to BASIC from a machine language routine. If you use the format CALL &XXXX,N then the value of variable N will automatically become the value contained in the 16-bit X register of the CPU when the machine language routine is exited *provided that the carry flag is set at the time of exiting!* You must be careful when exiting a machine language routine back to BASIC when you use this calling format *especially when you do not want the value of the variable to be altered by the ML routine!* Make sure, in such cases, that you clear the carry flag just prior to exiting. Otherwise, *whatever garbage is in the X register of the CPU will end up as an (unwanted!) value for your BASIC variable.* (I speak from experience having once spent some time tracing down a bug caused by my forgetting this point!)

Hybridizing (instead of going entirely to ML routines) does, of course, reduce the overall speed of a program. The BASIC interpreter introduces a lot of overhead in the process of passing control back and forth from it to a machine language routine. The amount of speed sacrificed depends on how many times the transfer between BASIC and ML takes place and other factors. However, even with such handicaps, speed improvements in the order of 10 to 1000 (over programming entirely in BASIC) are generally possible using the technique. You will get a good feel for this aspect if you are following along and actually implementing the routines presented in this series.

The Conversion

The first step in our goal of obtaining speedy line drawing capability is to convert the pixel control routine (that was labeled ON in the preceding article) into machine language. Did you take a try at doing this? Did you learn anything? I bet you did!

The accompanying listings show the routines I came up with using a hybridized approach. I split the process into two parts. One for handling the X coordinate and another for the Y coordinate. That is, I first passed the X value to the ML routine and massaged it into the desired form. When the first ML routine returns to BASIC, another CALL is used to pass the Y value to a second ML routine. This second ML routine combines the Y coordinate with the previously converted information and turns on the desired LCD pixel.

One reason for using this technique at this point was so that ML students could observe the transformation from BASIC to ML coding. That is, to illustrate how BASIC can be used to develop an algorithm (the ON routine in the preceding article) and then show how ML can be used to obtain speed enhancements. The ML routines presented here closely follow the structure of the original BASIC pixel ON routine. The labels used in the machine

Listing *ML Routine to Convert X-Coordinate Value into Display Buffer Address Value.*

PG	LC	B1	B2	B3	B4	B5	LABELS	MNEMONICS	COMMENTS
70	50	58	76				ON	LDYH #76	Place base address of LCD buffer into Y register.
70	52	5A	00					LDYL #00	Base address is 7600.
70	54	B5	00				COL	LDA #00	Load accumulator with zero.
70	56	AE	78	C0				STA 78C0	Stuff it into 78C0 to clear location used as F flag.
70	59	04						LDA XL	Pull X value (passed from BASIC) into the accumulator.
70	5A	B7	27					CPA #27	Compare it with 39 (decimal). I.e., is X<39?
70	5C	83	05					FBC 1020	Skip if have carry as that means XL=>39.
70	5E	D9						SLA	Multiply X value by 2.
70	5F	12						ADDA YL	Add (X*2) to base address value.
70	60	1A						STA YL	Store the result in Y.
70	61	8E	25					FB ROWSAV	Go to process pixel row value.
70	63	B7	4E				1020	CPA #4E	Compare X value with 78 (decimal). Is X<78?
70	65	83	09					FBC 1030	Skip if have carry as that means XL=>78.
70	67	B1	26					SUBA #26	Subtract 38 + complement of carry so have (X-39).
70	69	D9						SLA	Multiply the result by 2 so have (X-39)*2.
70	6A	12						ADDA YL	Add to base address value.
70	6B	1A						STA YL	Store the result in Y.
70	6C	FD	50					INYH	Form (X-39)*2+256 in Y.
70	6E	8E	18					FB ROWSAV	Go to process pixel row value.
70	70	B5	FF				1030	LDA #FF	Load accumulator with all ones.
70	72	AE	78	C0				STA 78C0	Stuff this into 78C0 to set location used as F flag.
70	75	04						LDA XL	Pull original X value (passed from BASIC) into A register.
70	76	B7	75					CPA #75	Compare it with 117 (decimal). Is X<117?
70	78	83	07					FBC 1040	Skip if have carry as that means XL=>117.
70	7A	B1	4D					SUBA #4D	Subtract 77 + complement of carry so have (X-78).
70	7C	D9						SLA	Multiply the result by 2 so have (X-78)*2.
70	7D	12						ADDA YL	Add to base address value.
70	7E	1A						STA YL	Store the result in Y.
70	7F	8E	07					FB ROWSAV	Go to process pixel row value.
70	81	B1	75				1040	SUBA #75	Form (X-117).
70	83	D9						SLA	Multiply by 2 so have (X-117)*2.
70	84	12						ADDA YL	Add to base address value.
70	85	1A						STA YL	Store in Y.
70	86	FD	50					INYH	Form (X-117)*2+256 in Y.
70	88	68	78				ROWSAV	LDUH #78	Set U to point to temporary storage location.
70	8A	6A	C1					LDUL #C1	Using 78C1 & 78C2 to store converted pixel column address.
70	8C	94						LDA YH	Get high portion into the accumulator.
70	8D	61						STAI (U)	Store into 78C1 with auto-increment of U pointer.
70	8E	14						LDA YL	Get low portion into the accumulator.
70	8F	2E						STA (U)	Store into 78C2.
70	90	F9						CLRC	Clear carry before departing subroutine.
70	91	9A						RTS	Exit back to BASIC to fetch pixel row (Y) value.

language version -- especially the reference to line numbers as ML labels -- are to assist the reader in following the conversion process.

The ML routines are highly commented so I will not use a lot of space discussing them in detail. You will have to do some studying of the listings if you really want to understand the routines in detail. I will, however, point out a few items that highlight differences between BASIC and ML programming.

Note, for instance, that at the start of the ML routine (labeled ON to correspond with the original BASIC version), the display base address &7600 is set up in CPU register Y. There is no need to assign it to a variable as was done in BASIC. The BASIC variable B can be eliminated from the ML version because advancing an address value by 256 is readily accomplished just by incrementing the high order byte of a two-byte address. Remember, too, that the X-coordinate value will be residing in CPU register X when this routine is entered, having been placed there by the BASIC CALL statement.

The ML routines use three designated memory locations as storage places. The flag F (in the BASIC program) is represented by the contents of location &78C0. A value of 00 here means the flag is cleared. A value of &FF represents a set state of the flag. Locations &78C1 and &78C2 are used to hold information contained in a CPU register. This information might be lost when the PC was interpreting BASIC statements, as it must do when

Listing *ML Routine to Convert Y-Coordinate Value to Display Buffer Address and Bit Value.*

PG	LC	B1	B2	B3	B4	B5	LABELS	MNEMONICS	COMMENTS
70	A0	68	78				ROW	LDUH #78	Set U to point to temporary storage location.
70	A2	6A	C1					LDUL #C1	Starts at 78C1, also uses 78C2.
70	A4	65						LDAI (U)	Fetch high portion into accumulator.
70	A5	18						STA YH	Store pixel column address (high part).
70	A6	25						LDA (U)	Fetch low portion into accumulator.
70	A7	1A						STA YL	Store pixel column address (low part).
70	A8	04						LDA XL	Fetch pixel row value (passed from BASIC).
70	A9	B7	04					CPA #04	See if less than 4.
70	AB	81	04					FBNC FLG	Skip ahead on non-carry as Y<4.
70	AD	81	04					SUBA #04	Subtract 4 so have Y=Y-4.
70	AF	50						INYL	If Y=>4 then advance display buffer pointer.
70	B1	40					FLG	INXL	Add 1 to pixel row value (0 to 6 becomes 1 to 7).
70	B2	B5	01					LDA #01	Set least significant bit in accumulator.
70	B4	42					LOOP	DEXL	Decrement adjusted row pixel value.
70	B5	8B	03					FBZ OUT	Jump out of loop when XL reaches zero.
70	B7	09						SLA	Shift bit in accumulator one position to the left.
70	B8	9E	06					RB LOOP	Go back to continue bit-positioning operation.
70	BA	0A					OUT	STA XL	Save positioned bit back in XL.
70	BB	A5	78	C0				LDA 78C0	Fetch F flag from its storage location.
70	BE	00						INA	Exercise the accumulator (in order to set flags).
70	BF	0F						DEA	Restore the accumulator (flags reflect original value).
70	C0	0A						STA XL	Save Y in XL.
70	C0	8B	06					FBZ 1080	If F flag was cleared, then skip ahead.
70	C2	04						LDA XL	Fetch positioned bit from XL.
70	C3	09						SLA	Rotate the bit into the upper nibble.
70	C4	09						SLA	Ditto.
70	C5	09						SLA	Ditto.
70	C6	09						SLA	Ditto.
70	C7	0A						STA XL	Restore positioned bit to XL.
70	C8	04					1080	LDA XL	Fetch positioned bit from XL.
70	C9	1B						ORA (Y)	Mesh the current bit with whatever is already in display.
70	CA	1E						STA (Y)	Update the display buffer. (At last, at last!)
70	CB	F9						CLRC	Clear the carry flag before departing.
70	CC	9A						RTS	Exit back to BASIC.

it goes back to pick up the Y-coordinate value.

The ML routines and the storage locations just discussed have been assigned to the area in RAM that is normally used to hold string variables A$ - O$. Avoid using a CLEAR directive when these ML routines are installed!

Loading and Experimenting

You can load the ML routines directly into memory from the accompanying listing. You can make up a BASIC program that uses POKE directives to set up the ML routines. Alternately, use a ML monitor program (such as Norlin Rober's LMD package sold by *PCM*) to store the code. Note that the first two columns in the listings provide the address. The next several columns indicate the actual machine code bytes that make up each ML directive. These are the hexadecimal values that must be placed into memory to form a sequence of ML directives.

Once the machine language routines are in memory, you need a BASIC routine that calls them and passes the appropriate X- and Y-coordinates. Here is how that can be done efficiently:

```
3000 "ON"CALL &70
     50,X:CALL &7
     0A0,Y:RETURN
```

To test your new capability you can use a simple input routine such as:

```
1000 "DOT"INPUT "
     X-COORD: ";X
1010 INPUT "Y-COO
     RD: ";Y
1020 CLS :GOSUB "
     ON"
1030 B$=INKEY$ :
     IF B$=""THEN
     1030
1040 GOTO 1000
```

This will allow you to specify display coordinates and observe that the appropriate pixel turns on.

You can combine the LINE routine and the input routines that started at lines 200 and 300 in the previous article to provide complete line drawing

capability. (If you are using the LMD package, you should renumber the input routines so they are above line number 999. Then they will not be mistaken as lines of source code. Label the start of each routine so you can access it from the RUN mode using a labeled directive.)

If you do this you will note that while lines can be drawn considerably faster than when the ON routine was entirely in BASIC, the process is still relatively slow. In order to obtain impressive speed it will be necessary to also convert the LINE routine from BASIC to machine language. Are you ready to tackle that task? Why not give it a try. You can compare your method with that presented in the next installment (which will conclude this series).

A RENUMBERING UTILITY

The Sharp PC-1500 and Radio Shack PC-2 are tremendously powerful devices in the hands of even a modest programmer. However, the recent introduction of the long awaited 16K RAM modules has reminded me of the major stumbling block in writing long application programs on these PCs: the lack of an easy to use renumbering utility. The purpose of this article is to describe an extremely fast "renumbering" routine written in only three lines of BASIC code.

Why do I consider the availability of a renumbering utility to be so important? The BASIC programming language itself is at fault. When a program branches through a GOTO or GOSUB it is necessary to specify a destination line number. Unless you are one of the very few who completely plans out programs in advance on paper, you will find yourself needing to decide what line number your program should jump to as you create code.

Can this be a problem? Suppose that you are going to frequently be calling an as yet unwritten subroutine that will, say, evaluate the polynomial P(x). You make a note on paper that the subroutine will be at line 2000. Every time you need this routine you must either remember it or else look up the the subroutine's starting line number. If you come back to update the program at a later date to modify it you will find yourself in the unpleasant situation of having to trace each subroutine reference in order to understand the program (unless you have otherwise documented the program in a thorough fashion). Things can rapidly get out of a hand in a long program where there may be dozens of GOTOs and GOSUBs. Remember, six months after the fact, a reference to line number 2000 might not do much to remind you that it handles P(x).

Fortunately, Sharp designed the PC-1500 with an unusually powerful feature that is lacking in many home computers: labeling. Any line may be labeled. All that is needed is to place the label in quotes immediately after the line number. You do not even have to place a colon between the label and the first statement in a line.

To jump to the line, you only need to state GOTO or GOSUB followed by the label of the desired line. This label reference must be enclosed in quotation marks. Thus, GOSUB 2000 might be replaced by GOSUB "P(x)" which is much more descriptive than GOSUB 2000. In fact, it is a fair example of self-documenting code.

A programmer has a virtually unlimited range of choices for labels since a label on the PC-1500 can be up to 73 characters in length and include blanks, upper and lower case letters, numerals and any other symbols except the quotation mark.

Since the computer supports this powerful feature, I strongly recommend that PC-1500 programmers use labels for all references to other routines and subroutines.

If the previously cited reasons are not sufficient to convince you of the worthiness of using labels, I can offer one more tantalizing appeal: the following program can renumber a 300 line program in about one minute. This program resides in just three lines of code. But, it only operates on programs wherein all GOTO and GOSUB references are by labels, not line numbers!

Here is the magic routine:

```
65024 END
65025 "L"B=STATUS
      2-STATUS 1,L
      =0
65026 IF PEEK B>25
      3THEN END
65027 L=L+10:POKE
      B,L/256,L-25
      6*INT (L/256
      ):B=B+3+PEEK
      (B+2):GOTO 6
      5026
```

The program may be executed by typing DEF L or RUN 65025. The program then renumbers all lines lower than 65024, assigning the first line number the value of 10 and using an increment between lines of 10. Note that the program does *not* change GOTO and GOSUB statements! Those statements *must* refer to labeled lines.

Here is an outline of how the renumbering program operates:

The first line is just a precaution that prevents other programs from accidently dropping down to the renumbering program.

Line 65025 finds and stores the address of the first byte of your program. This address is stored in variable B. It utilizes the STATUS function so the procedure works regardless of the amount of memory installed in your PC. This line also initializes the variable L which is used to store new line numbers.

The best way to understand the rest of the routine is to recall how programs themselves are stored in the PC-1500. Each BASIC line starts with a two-byte value, most significant byte first, that represents its line number. The third byte is equal to N+1 bytes, where N is the number of bytes in the body of the tokenized BASIC line. The next N bytes are those comprising the tokenized line. The byte after these N bytes (N+1) is the ASCII code for a carriage return (13). All line numbers must be greater than zero. The most significant byte of a line number must be less than 255. This latter restriction limits the programmer to using line numbers that are less than 65280. The last user line in the computer is terminated by the code 255.

The renumber utility ignores all line numbers above 65023. Thus, you can store other commonly used utility routines, such as a base converter, above the renumbering routine, etc. Note that line 65026 in the program terminates its own operation when a line number greater than 65023 is found.

Line 65027 increments the line number (variable L) by 10. It then POKEs it into the proper two bytes as indicated by variable B. Variable B is then incremented by N+4 so that it points to the address of the next BASIC line number. The program then goes back to line 65026 to determine whether it is to continue renumbering.

You may customize this program. The line increment set in line 65027 by the statement L=L+10 may be replaced with whatever increment value suits you. And, instead of setting L=0 in line 65025, you may set L equal to whatever offset you desire, such that the first line number becomes that offset value *plus* the line increment value.

Thanks for this utility procedure and routine go to: *Norman E. Beam, 21051 Gresham Street #103, Canoga Park, CA 91304.*

STAR TARGETS

This is a graphics game for the Radio Shack PC-2 and Sharp PC-1500. It was designed by *David A. Cloutier, Bullard Road, North Brookfield, MA 01535.* You need at least a 4K RAM module in your PC in order to use this program.

To play the game, imagine that you are a ball turret gunner in a space freighter. Your job is to eliminate all enemy fighters trying to destroy the freighter. When the title flashes on the screen, press ENTER *twice* and you will be all set for battle. Press the left arrow key to rotate your turret to the left. Use the right arrow key to rotate it to the right. (Note that the enemy will shift in the opposite direction.) Use the keys to position the enemy into the sights. When the sights start blinking, press the spacebar to fire!

```
3005:REM Star Tar
     gets
3010:CLEAR :CLS :
     DIM S$(0)*18
     ,SU$(0)*18,S
     D$(0)*18,C$(
     0)*18
3015:DIM D1$(0)*2
     2,D2$(0)*22,
     U$(0)*22
3020:S$(0)="120C1
     A2E2A2E1A0C1
     2"
3025:D1$(0)="0008
     2255361D6E10
     244B3A"
3030:SU$(0)="0906
     0F1517150F06
     09"
3035:D2$(0)="0002
     28120D3E0C32
     040012"
3040:SD$(0)="6418
     3C343C343C18
     64"
3045:U$(0)="63410
     000081C08000
     04163"
3047:GOSUB 3300
3050:T=TIME :WAIT
     0:G=60:
     GCURSOR G:
     GPRINT S$(0)
     ;
3060:C=0:RANDOM :
     R=RND 3:IF R
     =1THEN LET C
     $(0)=S$(0)
3070:IF R=2THEN
     LET C$(0)=SU
     $(0)
3080:IF R=3THEN
     LET C$(0)=SD
     $(0)
3090:BEEP 1,15,50
     :GR=(RND 8)-
     4:IF GR=0
     THEN LET GR=
     -4
3095:G=G+GR:IF G>
     155THEN LET
     G=0
3097:IF G<0THEN
     LET G=155
3098:GOSUB 3500:
     GOSUB 3400
3100:BEEP 1,20,50
     :CLS :
     GCURSOR G:
     GPRINT C$(0)
     :GOTO 3060
3300:WAIT 0:PRINT
     CHR$ 127;" S
     tar Targets
      V1.8 !";:
     CURSOR 25:
     PRINT CHR$ 1
     27
3302:IF INKEY$ =
     CHR$ 13THEN
     3302
3304:A=RND 200:B=
     RND 50
3306:GCURSOR 1:
     BEEP 1,A,B:
     GPRINT A,B;:
     GCURSOR 151:
     GPRINT A,B;
3308:IF INKEY$ <>
     CHR$ 13THEN
     3304
3310:CLS :PRINT "
     By David A.
     Cloutier":
     GOSUB 3850
3320:RETURN
3400:P=RND 7:IF P
     <>1THEN
     RETURN
3410:CLS :GCURSOR
     G:GPRINT "08
     04";C$(0);"0
     408";
3420:BEEP 5,7,200
     :P=RND 10:IF
     P<>1THEN
```

Program *Star Targets.*

```
      RETURN
3430: G1=78:G2=79
3440: BEEP 1,G2-70
      ,25:GCURSOR
      G1:GPRINT 25
      5-POINT G1:
      GCURSOR G2:
      GPRINT 255-
      POINT G2
3445: G1=G1-1:G2=G
      2+1
3450: IF G1<0OR G2
      >155THEN 345
      5
3452: GOTO 3440
3455: PRINT "You w
      ere blown to
       pieces!":
      GOSUB 3850
3460: PRINT "Score
       =";SC:GOSUB
      3850
3470: PRINT "Ships
       =";WS:GOSUB
      3850
3480: END
3500: GCURSOR 73:
      GPRINT "41";
      :GCURSOR 82:
      GPRINT "41"
3505: A$=INKEY$ :
      GOSUB 3590
3510: IF G<>74THEN
      RETURN
3520: CLS :GCURSOR
      73:GPRINT U$
      (0);:BEEP 1,
      8,100:A$=
      INKEY$
3530: IF A$=" "
      THEN 3600
3540: CLS :GCURSOR
      G:GPRINT S$(
      0);:BEEP 1,8
      ,200:C=C+1
3550: IF C=3THEN
      RETURN
3560: GOTO 3520
3590: IF A$=CHR$ 1
      2THEN LET G=
      G-5
3592: IF G<0THEN
      LET G=155
3595: IF A$=CHR$ 8
      THEN LET G=G
      +5
3596: IF G>155THEN
      LET G=0
3597: IF A$=" "
      THEN 3599
3598: RETURN
3599: A$=INKEY$ :
      GOTO 3597
3600: GCURSOR 73:
      GPRINT D2$(0
      );:BEEP 5,9,
      200
3610: CLS :GCURSOR
      73:GPRINT D1
      $(0);:BEEP 5
      ,20,50
3620: G1=72:G2=84
3630: GCURSOR G1:
      GPRINT (RND
      128)-1:
      GCURSOR G2:
      GPRINT (RND
      128)-1:BEEP
      1,G2,5
3640: G1=G1-4:G2=G
      2+4:IF G1<=0
      OR G2>=155
      THEN LET WS=
      WS+1:GOSUB 3
      700:GOTO 305
      0
3650: GOTO 3630
3700: SC=INT (SC+1
      00-(100*(
      TIME -T)))
3720: PRINT "Score
       =";SC:GOSUB
      3850
3730: PRINT WS;" e
      nemy ships d
      estroyed":
      GOSUB 3850
3840: GOTO 3860
3850: IF INKEY$ =
      CHR$ 13THEN
      3850
3855: IF INKEY$ <>
      CHR$ 13THEN
      3855
3857: IF INKEY$ <>
      CHR$ 13THEN
      3857
3860: CLS :RETURN
```

REDEFINING THE PC-1500 KEYBOARD

If you have POKEd around in the Sharp PC-1500 you may have noticed a little Japanese symbol appear in the display. This occurs when bit 2 of location &764E is set. (You can see it by entering: POKE &764E,PEEK &764E OR 4.) When the PC is in this mode, keyboard codes may be fetched from a user-built table. To do this, a pointer stored at &785D must be set to &80 when using a CE-150 or to &00 when using a CE-158 (RS-232) interface.

In this discussion, the use of a CE-150 will be assumed. For this case the user-built table must start at an address: &nn80. The high byte of this address (nn) is indicated to ROM by setting the pointer at &785E to (nn+1).

The first keyboard table contains 128 bytes. Sixty-four are used for the regular keyboard and 64 for when the shift key is utilized. Keys to be redefined must have ASCII codes greater than A0.

A second table holds GPRINT codes -- five per symbol -- that are used to define each character. This table is located at address &(nn+1)A0. The length of this second table depends on the number of keys redefined. It takes five bytes to define each key.

Thus, the directive POKE &785D,&80,&3A will indicate to ROM that a keyboard table starts at &3980 (going to &39FF) and that there is a GPRINT table starting at &3AA0.

(Keys having ASCII codes less than &80 will be fetched from the regular ROM table at addresses &FCA0 through &FE7F.)

The program shown in the accompanying listing capitalizes on this information. It *adds* a set of Greek alphabet characters to the repertoire of the PC-1500. Note that I said *adds* and not *substitutes*. All other capabilities remain. Thus, by a single keystroke, you may switch from one keyboard representation to another.

Actually, the program only redefines 35 keys. This provides 24 lowercase Greek letters, 10 uppercase Greek symbols (as the other 14 uppercase Greek characters are identical to English ones), and the apostrophe which Sharp did not bother to provide. To facilitate using these new symbols within BASIC programs, only lowercase keyboard characters were redefined.

The program consist of two parts. One POKEs the two necessary tables into the low portion of memory. The other is a little machine language routine that activates the Greek mode whenever desired.

Of course, these keyboard tables must be protected by an appropriate NEW statement to isolate them from the user's BASIC storage area. However, the program works with any RAM memory expansion (4K, 8K or 16K modules). This is

because the first available address is computed after examining the SOM (Start Of Memory) pointer that is maintained in address &7863.

Installation and Use

The first step you must take is to protect a section of low memory where the table will be stored. Do this by executing the command:

```
NEW(PEEK &7863+2)*256+336
```

(If you are using an 8K RAM module, then you could just enter: NEW &3B50.)

Next, load the entire BASIC program shown in the accompanying listing into memory.

Execute this program by typing RUN in the RUN mode. This causes the two tables to be placed into the protected area of low memory. After a few seconds the > prompt will come back up on the LCD. Now you can eliminate all of the BASIC lines *except for line 300*.

To save the keyboard tables on tape for future use, set up two variables by executing these statements:

```
A=(PEEK &7863+1)*256+128
B=(PEEK &7863+2)*256+335
```

Now place the tape unit in the record mode and execute the command:

```
CSAVE M "GREEK KB TABLE";A,B
```

(Again, if you are using an 8K RAM module, then use: CSAVE M "GREEK KB TABLE",&3980,&3B4F.)

Then, in the future you need only reserve the protected area in memory and load the keyboard table directly into memory using the command:

```
CLOAD M "GREEK KB TABLE";(PEEK &7863+2)*
256+128
```

You also need to restore line 300 (but you may renumber that as desired). Alternately, you may assign the contents of that line to a Reserved key:

```
(CALL(PEEK&7863+2)*256)
```

However, make sure you erase this assignation when the Greek table is not in memory!

That is all there is to it. Press the DEF/G key (or the Reserved key if you set it up as suggested) when you want to use Greek symbols. The Japanese

Diagram *Symbols Assigned to Numeric Keys.*

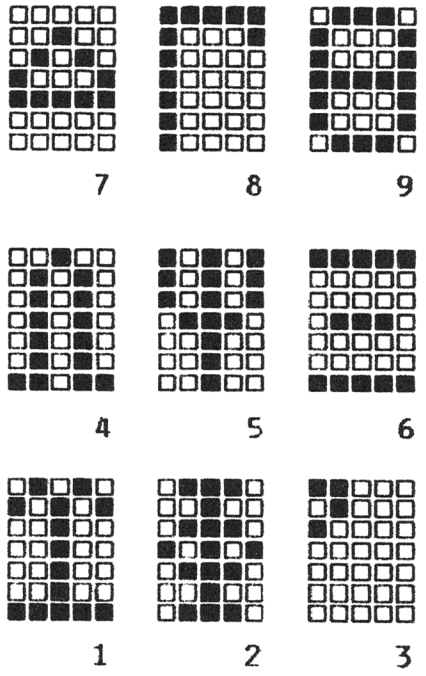
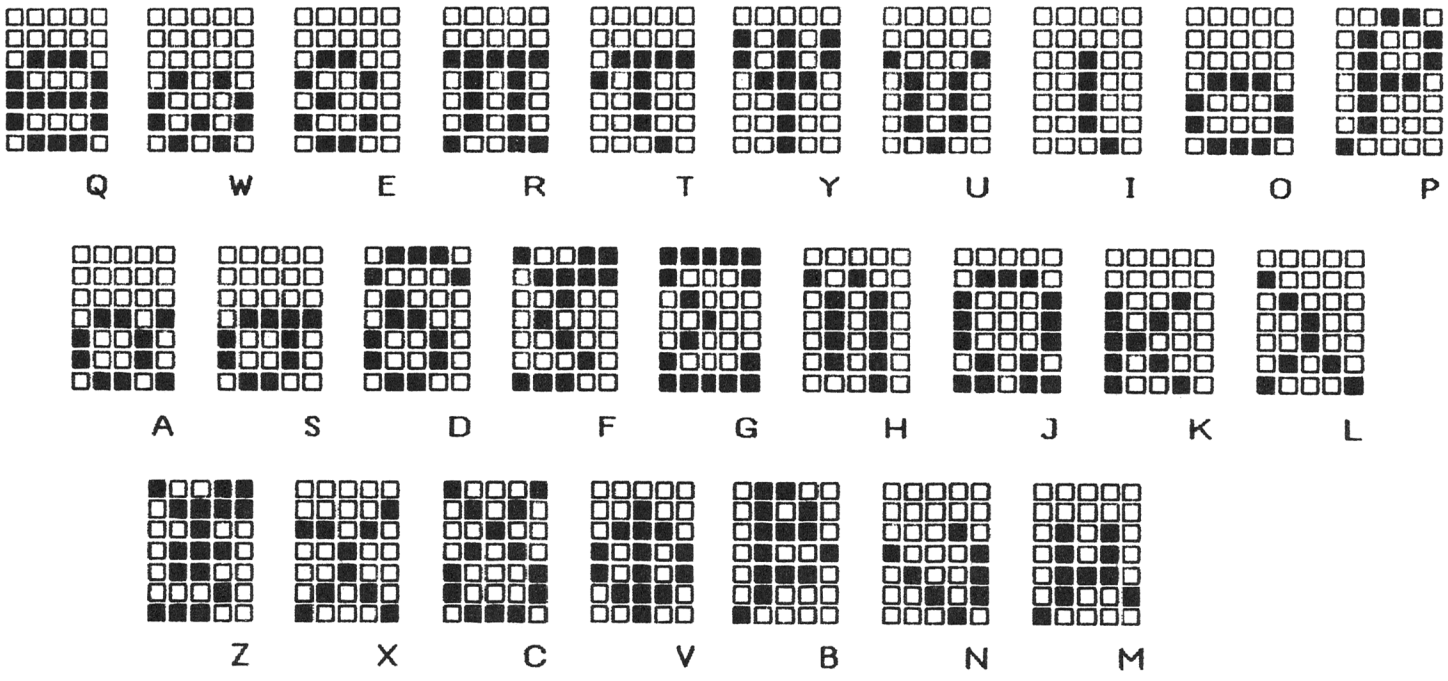


Diagram *Greek Symbols Assigned to Alphabetical Keys.*



symbol will come up on the display (to the left of the SMALL legend) to indicate that you are in this special keyboard mode. Remember these symbols are accessed using lowercase, so press the SHIFT key before pressing the desired alphabetical or numeric key. An accompanying diagram shows the Greek keyboard layout.

To switch back to the regular keyboard, press the SML key twice.

If you turn the PC off using the BREAK key, you will automatically quit the Greek mode. However, if the PC turns off by powering-down after the 7 minute time-out, you will remain in whatever mode you were using.

Of course, you may use these new Greek symbols just as you would any other characters: they can serve in labels, PRINT, PAUSE and INPUT statements, as well as file names.

Naturally the technique demonstrated here may be modified to enable your PC to communicate in whatever language you desire, be it Russian, Hebrew, or Sanscript. How about converting your PC to French, complete with accentuated vowels?

This powerful technique for customizing your PC keyboard was provided by: *Patrick L. Pollet, P.O. Box 144, UPM Box 298, Dhahran International Airport, Saudi Arabia.*

Program *Greek Keyboard*

```
1:REM greek alp
  habet keyboard
2:DATA &0B,&4E,&
  59,&01,&48,&38
  ,&35,&32,&09,&
  58,&57,&11,&53
  ,&1F,&2D,&2E
3:DATA &30,&4D,&
  55,&15,&4A,&37
  ,&34,&31,&0D,&
  28,&49,&16,&4B
  ,&4F,&4C,&29
4:DATA &19,&43,&
  45,&12,&44,&2F
  ,&2A,&2B,&20,&
  56,&52,&13,&46
  ,&50,&08,&3D
5:DATA &02,&5A,&
  51,&1B,&41,&18
  ,&1F,&0C,&0A,&
  42,&54,&14,&47
  ,&39,&36,&33
6:DATA &5B,&A0,&
  A1,&01,&A2,&A3
  ,&A4,&A5,&09,&
  A6,&A7,&21,&A8
  ,&0F,&2C,&2E
7:DATA &30,&A9,&
  AA,&25,&AB,&AC
  ,&AD,&AE,&0D,&
  3C,&AF,&26,&B0
  ,&B1,&B2,&3E
8:DATA &19,&B3,&
  B4,&22,&B5,&3F
  ,&3A,&3B,&5E,&
  B6,&B7,&23,&B8
  ,&B9,&1D,&40
9:DATA &02,&BA,&
  BB,&1B,&BC,&1A
  ,&1E,&1C,&5D,&
  BD,&BE,&24,&BF
  ,&C0,&C1,&C2
19:REM machine c
  ode routine to
   activate tabl
  es
20:DATA &B5,&80,&
  AE,&78,&5D,&A5
  ,&78,&63,&DD,&
  DD
21:DATA &AE,&78,&
  5E,&EB,&76,&4E
  ,&04,&9A,&38,&
  38
29:REM table gpr
  int codes of t
  he special cha
  racters
30:DATA &08,&10,&
  20,&44,&38,&06
  ,&08,&7E,&08,&
  06
31:DATA &02,&3C,&
  02,&7C,&00,&7F
  ,&01,&01,&01,&
  03
32:DATA &07,&08,&
  7F,&08,&07,&08
  ,&55,&7F,&55,&
  08
33:DATA &44,&24,&
  18,&24,&42,&30
  ,&48,&20,&48,&
  30
34:DATA &30,&48,&
  48,&38,&08,&40
  ,&3C,&10,&1C,&
  20
35:DATA &04,&38,&
  40,&38,&04,&5C
  ,&62,&02,&62,&
  5C
36:DATA &18,&14,&
  12,&14,&18,&40
  ,&7E,&01,&7E,&
  40
37:DATA &42,&41,&
  7E,&41,&42,&00
  ,&3C,&40,&00,&
  00
38:DATA &7C,&10,&
  28,&44,&00,&30
  ,&48,&48,&48,&
  30
39:DATA &42,&24,&
  18,&20,&40,&31
  ,&4A,&44,&4A,&
  31
40:DATA &28,&54,&
  44,&28,&00,&32
  ,&4D,&49,&31,&
  02
41:DATA &18,&24,&
  7E,&24,&18,&44
  ,&3C,&04,&7C,&
  44
42:DATA &41,&4A,&
  56,&23,&03,&40
  ,&3E,&09,&09,&
  06
43:DATA &41,&5A,&
  5E,&2B,&03,&38
  ,&54,&54,&54,&
  38
44:DATA &30,&48,&
  48,&30,&48,&40
  ,&3F,&15,&16,&
  08
45:DATA &08,&04,&
  3C,&44,&04,&63
  ,&55,&49,&41,&
  63
46:DATA &3E,&49,&
  49,&49,&3E,&41
  ,&49,&49,&49,&
  41
47:DATA &00,&05,&
  03,&00,&00
200:"D"A=(PEEK &78
  63+1)*256+&80
210:FOR I=0TO 127:
  READ M:POKE A+
  I,M:NEXT I
220:A=(PEEK &7863+
  2)*256
230:FOR I=0TO 19:
  READ ML:POKE A
  +I,ML:NEXT I
240:A=(PEEK &7863+
  2)*256+&A0
250:FOR I=0TO 174:
  READ M:POKE A+
  I,M:NEXT I
260:END
300:"G"CALL (PEEK
  &7863+2)*256:
  END
STATUS 1
                1758
```

FROM THE WATCH POCKET

I receive a lot of mail here at *PCN*. There is no way that I could respond to every inquiry. Over the years I have developed some basic rules or policies that I follow in dealing with this ongoing deluge.

The first is that I do not even look at letters that are rude and/or obnoxious. Should, by chance, one get by my secretarial staff, (and it is trained to remove letters from people with personal problems that have nothing to do with our line of work), I drop it into the wastebasket the moment it becomes evident that the intent of the letter is to vent someone's grief, hostility, anger or frustration. I am not a social or medical counselor nor a receptacle for abusive thoughts and ravings.

Many of the letters that I do review contain technical questions relating to subjects I have written about. If I am able to quickly respond to the questions by jotting down a brief reply (a few words to a few phrases - not a few paragraphs or pages!) and, *if the inquirer has had the courtesy of enclosing a self-address, stamped envelope*, then I often do forward a reply. (Not always, there is no guarantee, even with a S.A.S.E.)

But you can bet that if you do *not* include a self-addressed, stamped envelope, the chances of receiving a reply are approximately zero.

I do use comments received in letters to help shape the contents of *PCN*. For instance, if I receive a lot of letters from all around the country indicating that people are having difficulty using the CE-158 RS-232C interface, I assume there is a fair amount of interest in the subject. I thus select one or two articles dealing with that subject for publication.

Recently it seems that new purchasers of the CE-161 16K RAM module sometimes experience difficulty trying to use their Reserve keys. Despite the warnings published in the instruction manual, people are attempting to use these modules by entering NEW 0 after installation. The manual states to use NEW 256. Do what the manual says. If you don't, the system cannot set aside the proper storage space for the Reserve keys.

I frequently get letters asking about how to improve the performance of the tape cassette program/data storage and recall system used with the Sharp and Radio Shack PCs. The consensus seems to be that top quality tape is a great help. Can I name a brand that provides ultra-reliability? NO! I wouldn't attempt to do so, there are too many variables affecting the outcome. I can give some practical advice. Don't give too much credence to the word "certified" on a package of tape. Certified by whom, for what, under which conditions? A good rule of thumb is that if the brand you are using isn't working, then switch!

Can I name a tape recorder that works well with PCs? Sure. The Radio Shack Minisette-9, which is the model originally recommended by Radio Shack for use with their PCs, seems to do very well for the staff at *PCN*. Furthermore, we have received good reports about its use in this application from many other users. That does not mean you *have* to use a Minisette-9.

Do not expect perfection from any audio tape cassette recording system. They were never really meant to store computer data. It is not appropriate to attempt to store large amounts of critical data using these systems. The technology being used is such that one can expect to encounter an error in every 30 minutes or so of tape recorder use. It is wise to keep such limitations in perspective when you consider your applications!

From the Rumor Mills

It does appear as though 1984 will be the year of the notebook computer. Besides the ones already announced in *PCN*, the streets are rife with talk of entries from many more firms. It seems Casio may be one of the next to announce. Those already in the running are apparently planning on spending megabucks to promote their entries. Watch the magazines, TV and listen to radio for the barrage Sharp is expected to release touting the PC-5000.

I also expect to see a barrage of exciting new software offerings and PC add-ons in the coming year. Most of these will be from small companies that have figured the PC base is now big enough to take a gamble on.

Available Only by Prepaid Subscription for a Calendar Year Period (January - December). You are sent back issues for the calendar year to which you subscribe, at the time you enroll.

MC/VISA Phone subscriptions: (203) 888-1643

__ I an interested. Please send more information. I have:
__ a Sharp PC-1500 __ Radio Shack PC-2 __ ?_____

__ Enroll me as a 1984 Subscriber (Issue numbers 31-36). $18.00 in U.S. ($21.00 to Canada/Mexico. Elsewhere U.S. $30.00 payable in U.S. funds against a U.S. bank.)

__ Enroll me as a 1983-84 Subscriber (Issue numbers 21-36). $54.00 in U.S. ($63.00 to Canada/Mexico. Elsewhere U.S. $80.00 payable in U.S. funds against a U.S. bank.)

__ Enroll me as a 1982-84 Subscriber (Issue numbers 11-36). $78.00 in U.S. ($95.00 to Canada/Mexico. Elsewhere U.S. $120.00 payable in U.S. funds against a U.S. bank.)

__ Check here if paying by MasterCard or VISA. Please give credit card information below.

Orders must be accompanied by payment in full.

Name: _____

Addr: _____

City: _____ State: ____ Zip: _____

MC/VISA #: _____

Signature: _____ Exp. Date: _____

POCKET COMPUTER NEWSLETTER

P.O. Box 232, Seymour, CT 06483